IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicants:	Chi-Keung Luk, et al.	Examiner: Jennifer N. To
Serial No.:	10/029,699	Group Art Unit: 2195
Filed:	December 18, 2001	Docket No.: 200302164-1
Title:	Software Controlled Pre-Execution in a Multithreaded Processor	

APPEAL BRIEF UNDER 37 C.F.R. § 41.37

Mail Stop Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

This Appeal Brief is filed in response to the Final Office Action mailed February 5, 2007 and Notice of Appeal filed on April 5, 2007.

AUTHORIZATION TO DEBIT ACCOUNT

It is believed that no extensions of time or fees are required, beyond those that may otherwise be provided for in documents accompanying this paper. However, in the event that additional extensions of time are necessary to allow consideration of this paper, such extensions are hereby petitioned under 37 C.F.R. § 1.136(a), and any fees required (including fees for net addition of claims) are hereby authorized to be charged to Hewlett-Packard Development Company's deposit account no. 08-2025.

I. REAL PARTY IN INTEREST

The real party in interest is Hewlett-Packard Development Company, LP, a limited partnership established under the laws of the State of Texas and having a principal place of business at 20555 S.H. 249 Houston, TX 77070, U.S.A. (hereinafter "HPDC"). HPDC is a Texas limited partnership and is a wholly-owned affiliate of Hewlett-Packard Company, a Delaware Corporation, headquartered in Palo Alto, CA. The general or managing partner of HPDC is HPQ Holdings, LLC.

II. RELATED APPEALS AND INTERFERENCES

There are no known related appeals or interferences known to appellant, the appellant's legal representative, or assignee that will directly affect or be directly affected by or have a bearing on the Appeal Board's decision in the pending appeal.

III. STATUS OF CLAIMS

Claims 1 – 36 stand finally rejected. The rejection of claims 1 – 36 is appealed.

IV. STATUS OF AMENDMENTS

No amendments were made after receipt of the Final Office Action. All amendments have been entered.

V. SUMMARY OF CLAIMED SUBJECT MATTER

The following provides a concise explanation of the subject matter defined in each of the claims involved in the appeal, referring to the specification by page and line number and to the drawings by reference characters, as required by 37 C.F.R. § 41.37(c)(1)(v). Each element of the claims is identified by a corresponding reference to the specification and drawings where applicable. Note that the citation to passages in the specification and drawings for each claim element does not imply that the limitations from the specification and drawings should be read into the corresponding claim element or that these are the sole sources in the specification supporting the claim features.

The Brief Summary of the Invention provides the following summary in paragraph [0008]: The problems noted above are solved in large part using a processor (*e.g.*, a simultaneous multithreading processor) that can run a program in one thread (called the "main" thread) and at least a portion of the same program in another thread (called the "pre-execution" thread). In general, the concepts apply to any computer that can execute multiple threads of control that share common cache structures. The program in the main thread includes instructions that cause the processor to start and stop pre-execution threads and direct the processor as to which part of the program is to be run through the pre-execution threads. Preferably, such instructions cause the pre-execution thread to run ahead of the main thread in program order (*i.e.*, fetching an instruction from the program before the main thread fetches the same instruction). In this way, preferably any cache miss conditions that are encountered by the pre-execution thread are resolved before the main thread requires that same data. Therefore, the main thread should encounter few or no cache miss conditions, and overall performance is increased.

Claim 1

A computer system (Figs. 1-3: #90), comprising:

a processor (#100) capable of executing multiple threads (paragraphs [0019 – 0020]); and

a main system memory (#92) coupled to said processor (paragraph [0018]);

wherein said processor processes a program (#220) in a main thread that includes instructions (#222) which cause the processor to spawn a pre-execution thread in which only a portion, but not all, of the same program executes, said pre-execution thread runs concurrently with the main thread, but ahead of the main thread in program order to cause data to be cached before the main thread needs the data (paragraphs [0027 – 0029]).

Claim 4

The computer system of claim 1 wherein the pre-execution thread encounters a cache miss condition for a memory reference but the main thread does not encounter a cache miss condition when that same memory reference is processed by the main thread (paragraph [0029]).

Claim 10

A processor (Fig. 2: #100), comprising:

a fetch unit (#102) capable of fetching instructions from a plurality of threads (paragraph [0020]);

a program counter (#106) coupled to said fetch unit (paragraph [0020]);

an instruction cache (#110) coupled to said fetch unit (paragraph [0019]);

a data cache (#146) coupled to said instruction cache (paragraph [0019]);

wherein said processor processes a program (#220) in a first thread that includes instructions (#222) which cause the processor to spawn a second thread in which only a portion, but not all, of the same program also executes, said second thread runs concurrently with the first thread, but ahead of the first thread in program order so cache misses encountered by the second thread are resolved before the first thread encounters the cache misses (paragraphs [0027 – 0029]).

Claim 13

The processor of claim 10 wherein the second thread encounters a cache miss condition for a memory reference but the first thread does not encounter a cache miss condition when that same memory reference is processed by the first thread (paragraph [0029]).

Claim 19

A method of running a program in a processor (Fig. 4), comprising:

(a) inserting pre-execution thread instructions in the program (#252: paragraph [0033]);

(b) spawning a pre-execution thread when designated by the inserted instructions (#254: paragraph [0033]); and

(c) running said pre-execution thread concurrently with a main thread wherein both the pre-execution and the main threads include instructions from the same program, the pre-execution thread running ahead of the main thread and

containing only a portion of the instructions from the main thread so cache misses encountered by the pre-execution thread are resolved before the main thread encounters the cache misses (#256: paragraphs [0027 – 0029] and [0033 - 0034]).

Claim 24

The method of claim 19 further including creating sufficient slack in time for executing same instructions between the main thread and the pre-execution thread so the pre-execution thread has time to resolve cache misses before the main thread requests data previously subject to a cache miss (paragraph [0036]).

Claim 31

A processor (Fig. 2: #100), comprising:

a fetch unit (#102) capable of fetching instructions from a plurality of threads (paragraph [0020]);

a program counter (#106) coupled to said fetch unit (paragraph [0020]);

an instruction cache (#110) coupled to said fetch unit (paragraph [0019]);

a data cache (#146) coupled to said instruction cache (paragraph [0019]);

wherein, in a pre-execution thread, said processor pre-executes instructions (#220) from a main thread that are specified by said main thread so a cache miss encountered by the pre-execution thread is resolved before the main thread encounters the cache miss (paragraphs [0027 – 0029]).

VI. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

Claims 1-36 are rejected under 35 USC § 102(b) as being anticipated by USPN 5,812,811 (Dubey).

VII. ARGUMENT

The rejection of claims 1 – 36 is improper, and Applicants respectfully requests withdraw of this rejection.

The claims do not stand or fall together. Instead, Applicants present separate arguments for various independent and dependent claims. Each of these arguments is separately argued below and presented with separate headings and sub-heading as required by 37 C.F.R. § 41.37(c)(1)(vii).

Claim Rejections: 35 USC § 102

Claims 1-36 are rejected under 35 USC § 102(b) as being anticipated by USPN 5,812,811 (Dubey). Applicants respectfully traverse this rejection.

A proper rejection of a claim under 35 U.S.C. §102 requires that a single prior art reference disclose each element of the claim. See MPEP § 2131, also, *W.L. Gore & Assoc., Inc. v. Garlock, Inc.*, 721 F.2d 1540, 220 U.S.P.Q. 303, 313 (Fed. Cir. 1983). Since Dubey does not teach each element in the claims, these claims are allowable over Dubey.

Claim 1

Independent claim 1 recites numerous recitations that are not taught in Dubey. By way of example, claim 1 recites a pre-execution thread that runs "ahead of the main thread in program." Nowhere does Dubey teach a thread runs **ahead** of the main thread.

Dubey teaches future threads that "can be executed concurrently with the forking thread which continues to execute the sequence of instructions sequentially following the FORK" (see Dubey at 8: 60-65: emphasis added). Dubey, however, never discloses that the future threads run "ahead" of the forking thread. In fact, Dubey teaches that the fture threads execute at fork points that appear later in the trace order:

> A fork point refers to the position in the static instruction sequence where the available machine state is capable of starting a parallel execution of one or more sets of instructions which appear later

(but not immediately after the fork point) in the sequential trace order. (See Dubey at col. 14, lines 52-57: emphasis added).

Anticipation under section 102 can be found only if a single reference shows exactly what is claimed (see *Titanium Metals Corp. v. Banner*, 778 F.2d 775, 227 U.S.P.Q. 773 (Fed. Cir. 1985)). For at least these reasons, independent claim 1 and its dependent claims are allowable over Dubey.

As yet another example, claim 1 recites a pre-execution thread that runs ahead of the main thread in program order "to cause data to be cached before the main thread needs the data." Nowhere does Dubey teach or even suggest that a thread runs ahead of the main thread so data is cached before the main thread needs the data. Dubey teaches future threads that "can be executed concurrently with the forking thread which continues to execute the sequence of instructions sequentially following the FORK" (see Dubey at 8: 60-65). Dubey, however, never discloses that the future threads run ahead of the forking thread to cause data to be cached before the forking thread needs the data.

For a prior art reference to anticipate under section 102, every element of the claimed invention must be identically shown in a single reference (see *In re Bond*, 910 F.2d 831, 15 U.S.P.Q.2d 1566 (Fed. Cir. 1990)). For at least these reasons, independent claim 1 and its dependent claims are allowable over Dubey.

Claims 10, 19, 31

Independent claims 10, 19, and 31 recite numerous recitations that are not taught in Dubey. By way of example, each of these independent claims recites the following or a variation thereof: "so cache misses encountered by the second thread are resolved before the first thread encounters the cache misses." Nowhere does Dubey teach that a second thread runs ahead of the main thread so cache misses encountered by the second thread are resolved before the main thread encounters the cache misses.

Dubey teaches future threads that "can be executed concurrently with the forking thread which continues to execute the sequence of instructions sequentially following the FORK" (see Dubey at 8: 60-65). Dubey, however, never discloses that the future threads

run ahead of the forking thread so cache misses encountered by the future thread are resolved before the forking thread encounters the cache misses.

Anticipation is established only when a single prior art reference discloses each and every element of a claimed invention united in the same way (see *RCA Corp. v. Applied Digital Data Systems, Inc.*, 730 F.2d 1440, 1444 (Fed. Cir. 1984)). For at least these reasons, independent claim 10, 19, and 31 and their dependent claims are allowable over Dubey.

Claim 4

Claim 4 recites that the pre-execution thread encounters a cache miss condition for a memory reference but the main thread does not encounter a cache miss condition when that same memory reference is processed by the main thread. Dubey does not teach these recitations. The Examiner cites Dubey at column 15, lines 5- 40 for allegedly teaching these recitations. Applicants respectfully disagree.

Column 15, lines 5-40 in Dubey discuss that instruction sequences are fetched from the instruction cache for the main thread. This section then states that future threads are forked from the main thread. This section of Dubey, however, never states that the future threads are used to encounter cache misses before the main thread would encounter such cache misses. Dubey does state that the instructions are fetched for the main thread, but never states using the future thread to encounter cache misses, storing this reference, then using this stored reference so the main thread does not encounter the same cache miss.

For at least these additional reasons, Dubey does not anticipate claim 4.

Claim 13

Claim 13 recites that second thread encounters a cache miss condition for a memory reference but the first thread does not encounter a cache miss condition when that same memory reference is processed by the first thread. Dubey does not teach these recitations. The Examiner cites Dubey at column 15, lines 5- 40 for allegedly teaching these recitations. Applicants respectfully disagree.

Column 15, lines 5-40 in Dubey discuss that instruction sequences are fetched from the instruction cache for the main thread. This section then states that future threads are forked from the main thread. This section of Dubey, however, never states that the future threads are used to encounter cache misses before the main thread would encounter such cache misses. Dubey does state that the instructions are fetched for the main thread, but never states using the future thread to encounter cache misses and then the main thread avoids the cache miss when the same memory reference is processed by the main thread.

For at least these additional reasons, Dubey does not anticipate claim 13.

Claim 24

Claim 24 recites creating sufficient slack in time for executing same instructions between the main thread and the pre-execution thread so the pre-execution thread has time to resolve cache misses before the main thread requests data previously subject to a cache miss. Dubey does not teach such recitations.

Instead of addressing the actual recitations in claim 24, the Examiner states "As per claims 20, 22-24, and 26-29, they are rejected for the same reasons as claims 2-9" (see final OA at p. 6). Claims 2-9, however, do not recite creating "sufficient slack in time" such that "the pre-execution thread has time to resolve cache misses before the main thread requests data previously subject to a cache miss." The citations of the Examiner in Dubey are not even related to creating slack in time as recited in claim 24.

The Examiner has failed to establish a prima facie case for rejecting claim 24. Applicants have reviewed Dubey but cannot find any location that teaches creating "sufficient slack in time" such that "the pre-execution thread has time to resolve cache misses before the main thread requests data previously subject to a cache miss."

For at least these additional reasons, Dubey does not anticipate claim 24.

Response to Examiner's Arguments

The Examiner makes several arguments (see final OA beginning at p. 9). Applicants address some of these arguments not previously addressed herein above.

First, the Examiner argues that "Dubey teaches the future threads ahead of the forking thread" (see p. 9 point c). This statement is completely inaccurate. Nowhere does Dubey state that the future threads execute "ahead" of the forking thread. By contrast, expressly teaches that these threads execute concurrently:

> These future threads can be executed concurrently with the forking thread which continues to execute the sequence of instructions sequentially following the FORK" (see Dubey at 8: 60-65: emphasis added).

Dubey never teaches future threads executing ahead of the forking thread. The Examiner makes this conclusion without supportive teaching in Dubey.

Second, the Examiner states "The intended result/use of this limitation is to cause data to be cached before the forking thread needs data, and to resolve cache misses encountered by the future thread …" (see final OA at p. 9 point c). This statement is not supported in Dubey. Nowhere does Dubey state or even suggest that the forking thread executes a **same program** (as recited in the independent claims) to cache data before the main thread needs the data. The Examiner is stating conclusions and inferences not supported in the reference itself. Dubey itself, however, never discloses that the future threads run ahead of the forking thread so cache misses encountered by the future thread are resolved before the forking thread encounters the cache misses.

Third, the Examiner argues that the structure in Dubey is substantially identical to the structure of the claims. Applicants strongly disagree. Dubey does teach a future thread and a cache. However, nowhere does Dubey disclose that the future threads run ahead of the forking thread so cache misses encountered by the future thread are saved and then resolved before the forking thread encounters the cache misses.

Fourth, the Examiner argues that the recitations claimed but not shown in Dubey are "inherent" in Dubey. Applicants disagree. As stated in MPEP 2112 (no emphasis

added), in "relying upon the theory of inherency, the examiner must provide a basis in fact and/or technical reasoning to reasonably support the determination that the allegedly inherent characteristics necessarily flows from the teachings of the applied prior art."

Dubey teaches multiple threads but never states or even suggests that that the future threads run ahead of the forking thread so cache misses encountered by the future thread are saved and then resolved before the forking thread encounters the cache misses. Without such a statement or even suggestion, the recitations of the independent claims cannot "necessarily flow from the teachings" in Dubey.

CONCLUSION

In view of the above, Applicants respectfully request the Board of Appeals to reverse the Examiner's rejection of all pending claims.

Any inquiry regarding this Amendment and Response should be directed to Philip S. Lyren at Telephone No. 832-236-5529. In addition, all correspondence should continue to be directed to the following address:

Hewlett-Packard Company
Intellectual Property Administration
P.O. Box 272400
Fort Collins, Colorado 80527-2400

Respectfully submitted,

/Philip S. Lyren #40,709/

Philip S. Lyren
Reg. No. 40,709
Ph: 832-236-5529

VIII. Claims Appendix

1. A computer system, comprising:

a processor capable of executing multiple threads; and

a main system memory coupled to said processor;

wherein said processor processes a program in a main thread that includes instructions which cause the processor to spawn a pre-execution thread in which only a portion, but not all, of the same program executes, said pre-execution thread runs concurrently with the main thread, but ahead of the main thread in program order to cause data to be cached before the main thread needs the data.

2. The computer system of claim 1 wherein said instructions that cause the pre-execution thread to be spawned include a start instruction which causes a pre-execution thread to start and a stop instruction which causes the pre-execution thread to stop.

3. The computer system of claim 2 wherein said start instruction includes a value designating the location in the program where the pre-execution thread is to start running.

4. The computer system of claim 1 wherein the pre-execution thread encounters a cache miss condition for a memory reference but the main thread does not encounter a cache miss condition when that same memory reference is processed by the main thread.

5. The computer system of claim 1 wherein said processor determines whether sufficient hardware resources are available before spawning said pre-execution thread.

6. The computer system of claim 1 wherein said processor ignores exception conditions generated during the pre-execution thread.

7. The computer system of claim 1 wherein said processor does not permit a store instruction in the pre-execution thread to modify main system memory contents.

8. The computer system of claim 1 wherein said instructions that cause the pre-execution thread to be spawned include a value that informs a program counter where to begin executing the pre-execution thread.

9. The computer system of claim 1 further including a buffer into which pre-execution thread stores data is written to make such store data available to pre-execution thread load instructions.

10. A processor, comprising:

a fetch unit capable of fetching instructions from a plurality of threads;

a program counter coupled to said fetch unit;

an instruction cache coupled to said fetch unit;

a data cache coupled to said instruction cache;

wherein said processor processes a program in a first thread that includes instructions which cause the processor to spawn a second thread in which only a portion, but not all, of the same program also executes, said second thread runs concurrently with the first thread, but ahead of the first thread in program order so cache misses encountered by the second thread are resolved before the first thread encounters the cache misses.

11. The processor of claim 10 wherein said instructions that cause a second thread to be spawned include a start instruction which causes the second thread to start and a stop instruction which causes the second thread to stop.

12. The processor of claim 11 wherein said start instruction includes a value designating the location in the program where the second thread is to start running.

13. The processor of claim 10 wherein the second thread encounters a cache miss condition for a memory reference but the first thread does not encounter a cache miss condition when that same memory reference is processed by the first thread.

14. The processor of claim 10 wherein said processor determines whether sufficient hardware resources are available before spawning said second thread.

15. The processor of claim 10 wherein said processor ignores exception conditions generated during the second thread.

16. The processor of claim 10 wherein said processor does not permit a store instruction in the second thread to modify data cache contents.

17. The processor of claim 10 wherein said first and second threads execute same instructions but the second thread executes the same instructions before the first thread to resolve cache misses before the first thread requests data previously subject to a cache miss.

18. The processor of claim 10 further including a buffer into which pre-execution thread store data is written to make such store data available to pre-execution thread load instructions.

19. A method of running a program in a processor, comprising:

(d) inserting pre-execution thread instructions in the program;

(e) spawning a pre-execution thread when designated by the inserted instructions; and

(f) running said pre-execution thread concurrently with a main thread wherein both the pre-execution and the main threads include instructions from the same program, the pre-execution thread running ahead of the main thread and containing only a portion of the instructions from the main thread so cache misses encountered by the pre-execution thread are resolved before the main thread encounters the cache misses.

20. The method of claim 19 further including stopping the pre-execution thread when designated by the inserted instructions in (a).

21. The method of claim 19 further including copying register contents associated with the main thread to registers used by the pre-execution thread.

22. The method of claim 19 further including determining whether sufficient hardware resources are available to spawn the pre-execution thread.

23. The method of claim 19 further including ignoring all exception conditions generated during the pre-execution thread.

24. The method of claim 19 further including creating sufficient slack in time for executing same instructions between the main thread and the pre-execution thread so the pre-execution thread has time to resolve cache misses before the main thread requests data previously subject to a cache miss.

25. The method of claim 19 further including copying the contents of at least one register to memory to make such contents available to the pre-execution thread.

26. The method of claim 19 further including writing pre-execution store data into a buffer that can be accessed by a pre-execution load instruction.

27. The method of claim 19 further including not permitting any store instruction in the pre-execution thread to modify memory contents.

28. The method of claim 19 wherein (a) includes inserting a start instruction which causes the processor to start the pre-execution thread.

29. The method of claim 19 wherein the start instruction specifies a location in the program at which the pre-execution thread is to start running.

30. The method of claim 19 wherein (a) includes inserting a stop instruction which causes the processor stop the pre-execution thread.

31. A processor, comprising:

a fetch unit capable of fetching instructions from a plurality of threads;

a program counter coupled to said fetch unit;

an instruction cache coupled to said fetch unit;

a data cache coupled to said instruction cache;

wherein, in a pre-execution thread, said processor pre-executes instructions from a main thread that are specified by said main thread so a cache miss encountered by the pre-execution thread is resolved before the main thread encounters the cache miss.

32. The processor of claim 31 wherein the pre-execution thread is caused to be spawned to pre-execute said instructions by an instruction in the main thread.

33. The processor of claim 31 wherein the pre-execution thread spins on a variable that is set to a predetermined value by the main thread when there are instructions to pre-execute.

34. The processor of claim 31 wherein the processor ceases pre-executing instructions when a program counter is encountered that exceeds a range.

35. The processor of claim 31 wherein the processor ceases pre-executing instructions when the main thread catches up to the pre-executing instructions.

36. The processor of claim 31 wherein the processor ceases pre-executing instructions when the number of pre-executing instructions exceeds a limit.

IX. EVIDENCE APPENDIX

None.

X. RELATED PROCEEDINGS APPENDIX

None.